## REMARKS AND REQUEST FOR INTERVIEW

This paper responds to the Office Action mailed January 20, 2010 for the above application in which claims 144, 145, 155-158, 160, 164-165, 167-172 and 1615 are pending. Reconsideration of the application and claims in light of the following is requested.

***Request for Interview***

Applicant respectfully requests an interview with the Examiner to discuss the instant paper and application. Applicant's representative will be contacting the Examiner in this regard in the very near future to schedule the interview, hopefully to occur contemporaneous with the Examiner taking up the instant application for further examination based upon this paper.

***Response to Objection to Claim 144***

With respect to the phrase "being each being," claim 144 has been amended to remove the first "being" thereby fixing the grammar.

With respect to the "defined by a configuration whereby" language, to respond to the Examiner's question, it is respectfully noted that there is no structural difference between this phrase and something being configured as specified. Rather, this language addresses the specific point raised in paragraph 27 of the September 9, 2009 Office Action which stated:

> 27. Applicant argues that each of the cited references does not disclose processing resources of multiple types. In making each of the arguments, applicant relies on specific definitions of what a "type" of a processing resource is. However, applicant fails to point out where these definitions are required by the claim language. The examiner has reviewed the claims and is unable to find such a requirement. The examiner therefore concludes that applicant is merely arguing for subject matter that is not actually claimed. The examiner respectfully submits that such subject matter must be explicitly recited in the claim language in order to distinguish the claims over the prior art.

Specifically, this language was intended to explicitly bring the claim in line with applicant's arguments as called for by the Examiner. There is no magic to the terms "first type" and "second type" – they are merely labels for processing resources that are fundamentally different. What makes a processing resource a "first type" or "second type" processing resource is conformance with the characteristics specified in the claims for that "type" (i.e. it is configured

to operate (and will operate) in a particular way). Thus, a processing resource that is configured so that it "will receive a non-delegated instruction from memory that it can execute or delegate for execution elsewhere" is by definition a "first type" processing resource of the claim. A device lacking this characteristic is not a "first type" processing resource.

Similarly, what makes certain processing resources each a "processing resource of the second type" are those characteristics specified in the claims for the "second type." Thus, processing resources that have the two different specific characteristics specified by the claim (i.e. they "only receive and execute instructions that were delegated to them by processing resources of the first type" and they are "shared by at least two elemental processing resources of the first type") are by definition "second type" processing resources of the claims. A device that does not have both these characteristics is not a "second type" processing resource of the claims and one that lacks one or both these characteristics is not a "second type" processing resource of the claims.

Note also that the characteristics differentiating the two "types" are mutually exclusive because, by definition, "first type" processing resources only receive non-delegated instructions and either execute or delegate them, whereas "second type" processing resources only execute instructions delegated to them by "first type" resources **AND** must be shared by at least two "first type" processing resources.

These distinctions will be further elaborated on relative to the cited art below in connection with addressing the art-based rejections.

***Response to Claim Rejections Under 35 U.S.C. §112, ¶1***

Claims 144-145, 155-158, 160, 164-165 and 167-172 were rejected under 35 U.S.C. §112, ¶1 because certain claim language allegedly lacked written description.

First, the Office Action asserts that the claim 144 language stating "the processing resources of the second type each being defined by being configured to only receive and execute instructions that were delegated to them by processing resources of the first type" lacks written description support. Applicant disagrees and specifically directs the Examiner to the text found in paragraph [0055] and [0057] of the published application.

Second, the Office Action asserts that limitation of claim 1615 stating "" lacks written description support. Applicant disagrees and directs the Examiner to at least the text found in paragraphs [0056], [0057] and [0177] of the published application.

Text from these paragraphs are provided in this paper below.

Specifically, for the convenience of the Examiner, paragraphs [0055] thru [0057], [0107] and the contents of paragraphs [0022], [0023], [0025], [0027], [0053] thru [0057], [0062], [0063] [0066], [0077], [0100], [0104], [0105], [0177] that individually or collectively provide context and/or support are reproduced below, with each being preceded by the paragraph number in which they appear in the published application and with certain relevant language bolded to ease review. It should be understood by the Examiner that the associated figures referenced in (or pertinent to) those passages are intended to be included in the citation and that the bolding is not intended to be exclusive of the other text in that paragraph or other paragraphs not provided.

**[0022]** FIG. 9 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing **a resource delegation process in an integer processing unit in which the integer processing unit constructs and delegates a request to other processing resources**;

**[0023]** FIG. 10 is of a block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a form of **a delegation request constructed at a requesting integer processing unit and directed to a mathematical processing unit**;

**[0025]** FIG. 12 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of **request handling in a router of a processor routing requests between the requesting processing resources and servicing processing resources**;

**[0027]** FIG. 14 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of **routing a result from a target processing resource to delegating processing resources**

**[0053]** FIG. 2 is of a functional block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an architecture of Processor 100 of the present invention. **Processor 100 comprises Integer Processing Units ("IPUs") 102, 104, 106, 108, a Router 110, a secondary (i.e., level two) cache memory ("L2 cache") 112 and a Mathematical Processing Unit ("MPU") 114.** The IPUs are disposed in communication with the L2 cache and the MPU via the Router along with System Buses 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148. **In this exemplary architecture, four IPUs 102, 104, 106, 108 are collaborating with single MPU 112 and L2 cache 114 (i.e., a single MPU 112 and L2 cache 114 are shared by four IPUs 102, 104, 106, 108). Alternatively, the number of IPUs, MPUs and L2 caches may vary in other embodiments depending on the requirement of specific applications.**

**[0054]** The configuration of Processor 100 will depend on the context of system deployment. Factors such as, but not limited to, the desired throughput of a processor, capacity and/or location of the underlying system, nature of desired system deployment (e.g., portable, desktop, server, and/or the like environments) may affect deployment requirements and configuration. **It should be further noted that disparate processing resources may be employed in various quantities. Processing resources may include digital signal processing units (DSPs), graphic processing units (GPUs), IPUs, input/output controller processing units, memory management units (MMUs), MPUs, processing cache memory, vector processing units (VPUs), and/or the like. For example, in certain embodiments such as 3D animation rendering, floating-point throughput is desirable and Processor 100 may be configured with a single IPU and four MPUs interconnected by a Router. In yet another example embodiment, such as for a high traffic web server, handling multiple web requests simultaneously may require Processor 100 configured with 16 IPUs, a single MPU, and larger processing cache interconnected by a Router.** Further, in certain environments, processing resources may employ fewer or greater than 32-bit addressing and instruction pathways. For example, for environments where working with large datasets, 64-bit instruction and addressing may be employed for any of the processing resources.

**[0055] It should be noted that although processing resources on Processor 100 may be external to each other, the architecture and arrangement of the processing resources makes them intra-chip dependent and interactive.** This architecture is significantly different from current and prior art designs. Rather than merely duplicating the functionality of an existing, discrete, monolithic microprocessor design for a system to achieve a multiprocessor system a radical rethinking and implementation is exhibited by Processor 100 of the present invention. **In other words, elemental processing resources (e.g., IPUs, MPUs, L2 caches), which individually may not be capable of servicing an entire instruction set, are dependant on one another. These intra-dependent processing resources, together, service an instruction set more efficiently. Further, through instruction set design and intra-chip networking via a Router, these intra-dependent processing resources are capable of identifying other processing resources to which they may delegate instruction signals that they themselves may be incapable of servicing.** This intra-dependent, networked, processing resource, design chip architecture is more efficient than current and prior art monolithic designs. For example, prior art systems may combine two Intel Pentium chips in a system for extra integer performance. However, such a system will be inefficient as such monolithic chip duplication will also duplicate extra mathematical processing logic, memory management logic, etc., which may go underutilized. Furthermore, such prior art designs depend on processor intensive and time expensive operating system scheduling to make use of the added processing facilities. **Processor 100 of the present invention does not suffer from these limitations. It may employ numerous IPUs to enhance integer processing throughput, and itself may delegate**

**instruction processing to the proper processing resources without duplicating underutilized portions of a chip.**

**[0056] IPUs 102, 104, 106, 108 are the general integer and logic processing units of Processor 100 and each of the IPUs is responsible for running a separate program thread (i.e., a part of a program that can be executed independently of other IPUs). Furthermore, each of the IPUs may run independent processes and program software.** In one embodiment, each IPU may be a simple 32 bit microcomputer with 8.times.32 bit data registers (D0-7), 8.times.32 bit address registers (A0-7), a 32-bit instruction pointer (IP) and a 32 bit return link pointer (RP). Address register A7 may be used as a stack pointer. **Alternatively, each IPU may be a 32-bit or 64-bit microcomputer.** A 32-bit IPU may have a bank of 16 general purpose registers of 32 bits, organized as 8 address registers and 8 data registers. In a 32-bit implementation of Processor 100, 64 bit operations may take an extra cycle (i.e., double precision operations). A 64-bit IPU may have a bank of 16 general purpose registers of 64-bits, organized as 8 address registers and 8 data registers.

**[0057] Each of IPUs 102, 104, 106, 108 is configured to execute the instructions of a thread and/or process and perform relatively simple calculations of the instructions such as add, subtract, basic logic (e.g., Boolean operations) and/or integer calculations. Each of the IPUs are further configured, upon decoding an instruction while executing the instructions of the thread and/or process, to send calculation requests along with data and opcode of the instruction to MPU 114 for relatively complex calculations such as multiplication, division and/or floating point calculations. Alternatively, the IPUs may be configured to send the instructions themselves that require complex calculations to the MPU without decoding the instructions.** Similarly, each of the IPUs may also be configured, while executing the instructions of the thread and/or process, to send access requests to L2 cache 112 for accessing data and/or instructions stored in the L2 cache.

**[0062] MPU 114, one of the other shared resources in Processor 100, is a complement processing resource for IPUs 102, 104, 106, 108 and is configured to perform relatively complex calculations of instructions such as multiply, divide and/or floating point calculations upon receiving requests from the IPUs. The MPU is configured to receive calculation requests from the IPUs with decoded opcodes and data via Router 110 along with data bus 140. Alternatively, the MPU may be configured to receive and execute encoded instructions with complex calculations forwarded from the IPUs.** The calculation results are delivered via return data bus 148 to the requesting IPUs. The MPU may be lightly pipelined to deal with multiple requests from the IPUs before the completion of a request from an IPU (i.e., accepts new requests from the IPUs while others are in progress). For example, the MPU is pipelined with separate short pipes for floating point add/subtract, and integer/floating point multiply/divide operations. A state machine implements multi-sequence iterative functions such as integer/floating point divide, square root, polynomials and sum

of products. These in turn may be used to efficiently implement more complex functions.

**[0063] In one non-limiting implementation of Processor 100, a single MPU (e.g., MPU 114) is shared by multiple IPUs (e.g., IPUs 102, 104, 106, 108) using the silicon area more effectively when compared to a conventional processor with a similar size.** The streamlined architecture of the IPU structure (e.g., 32 bit microcomputer) makes it practical to incorporate several of the IPUs on a single chip. Current conventional designs become even more inefficient in contrast to Processor 100 of the present invention as the effective area of silicon required becomes even greater when such conventional processors are duplicated and employed as monolithic structures in parallel.

**[0066]** If the arithmetic operation required is more complex than integer add, subtract or compare, or floating point sign/magnitude compare, the operation may not be executed by the IPU. Instead the B and C parameters of the complex arithmetic operation are loaded into the operand pipeline registers and **a request issued for access to the shared MPU**, which can accept a new operation into its pipeline or pipelines every clock cycle. If more than one IPUs request MPU service in the same cycle, the request from the IPU which is running at the highest priority is granted. The requests from the IPUs are pipelined with pipeline input registers. A decoder decodes the requests determining the requesting IPU number. The decoded number of a requesting IPU may be added to the output result at the pipeline output register. In the execute pipeline phase, **an IPU may issue a request to the Router to connect to the shared MPU to execute, for example, a complex math instruction.** On the next free cycle, the IPU may write the function parameters to the input registers of the MPU. If the instruction requires double precision, the writing of the function parameters may take two cycles. **When the MPU has calculated the result, it returns the result on the dedicated return bus to all IPUs, with the IPU number of the corresponding request so the correct IPU will accept it.** If the result is double precision, this may take two cycles as well.

**[0077]** Each of Branched Data Buses are connected to Router 110 for transmitting a request to the shared resources (e.g., L2-cache 112 and MPU 114). **For example, the operand and/or data of an instruction that requires a complex calculation may be sent to the shared resources** (e.g., MPU 114) before the ALU of the IPU performs the calculation.

**[0100] In every clock cycle, every IPU may send a request to one of the shared resources (i.e., L2 cache 112 and MPU 114).**

**[0104]** FIG. 9 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure **providing a resource delegation process in IPUs 102, 104, 106, 108 of Processor 100 in which a requesting IPU constructs and delegates a request to other processing resources (e.g., L2 cache 112 and/or MPU 114).**

**[0105] The general operation iterative flow for a delegation process in one of an IPU (e.g., IPU 102) may be summarized as: (1) the constructing of a request if necessary to a delegated resource, (2) routing the request, and (3) obtaining a response from the delegated resource at the requesting unit.** At some point, an instruction is fetched for the requesting IPU 903. Upon obtaining an execution-instruction, a determination is made by the IPU if the instruction is intended for execution on one of the other processing resources (e.g., L2 cache 112 and/or MPU 114) based on the decoding of the opcode of the instruction 905. For example, if the opcode indicates that an adding or subtracting calculation is required, and no information is required from the external cache (i.e., L2 cache 112), the instruction then may be executed in the IPU without delegation 907. **If, however, the opcode indicates that at least one of the other processing resources is required to service the instruction** (e.g., a multiplication or division calculations, or information from the L2 cache is required for the execution), then a request may be constructed by the IPU 909. For example, in constructing the request, the IPU may combine several information such as the IPU number, decoded opcode, values from the status registers and priority bit status registers (e.g., lock/unlock/interrupt hardware priority bit). Subsequently, the IPU provides the constructed request to Router 110 911. Upon providing the request to the Router, the IPU is put to sleep and waits for a reply or for a request for the IPU to wake 913.

**[0177]** FIG. 25 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a wait-on-semaphore instruction facility. A wait-on-semaphore instruction is an atomic operation for a processing cache (e.g., L2 cache 112). At some point, a wait-on-semaphore instruction is fetched 2501 and **used to claim exclusive use of some shared resource** or memory 2503. Upon interpreting a wait-on-semaphore instruction operation 2505, the data optimizing cache will subtract "1" from a semaphore variable in the L2 cache 2507. If the processing cache determines that the semaphore variable is negative 2509, an operating system trap-call to wait on the queue is issued 2513. **This trap-call causes rescheduling so that other program threads/processes may wait for execution of a thread to release its semaphore on a particular IPU.** If the semaphore variable is not negative 2509, the processing cache may continue to operate 2511.

It is respectfully submitted that in the claims are all supported and withdrawal of the Section 112 rejections regarding written description is respectfully requested.

***Response to Section 103 Rejections***

Claims 144, 155-156, 164-165 and 167-170 have again been rejected for obviousness over a combination of the previously cited Bonola and Bridges references. Claim 144-145 and 171-172 have been rejected again for obviousness over a combination of the previously cited

Butterworth and Bridges references. Claim 157-158, 160 and 1615 have again been rejected for obviousness over a combination of the previously cited Bonola, Bridges and Mohamed art.

Applicant respectfully traverses the obviousness rejections for the following reasons.

In responding to Applicant's prior arguments, the Office Action asserts at p. 14-15:

> Applicant alleges that none of Bonola, Bridges or Butterworth discloses processors of a second type. In making these arguments, however, applicant does not reference any claim language or any portion of the prior art references. It is impossible for the examiner to determine on what basis applicant is making these allegations. It is the examiner's position that, as outlined in the rejection above, the master/slave configurations of Bonola, Bridges and Butterworth constitute the claimed invention in that they teach different processing elements that are used to process different types of operations (and wherein instructions are delegated from master processors to slave processors).

In the interest of brevity, since the rejections are again the same, Applicant respectfully incorporates the prior two responses regarding Bonola, Bridges, Butterworth and Mohamed by reference as if fully set forth herein as augmented by the following.

As previously asserted, each of the cited references is significantly different from what is claimed in the instant application. Specifically, all of the cited references lack, among other things, "second type" processing resources. Lacking such "second type" resources, none of the references, taken alone or in combination, disclose or even hint at the sharing of such resources as specified in the claims, which is required for performing the claimed methods.

As requested by the Examiner, since Applicant persists in the previously stated positions, the prior arguments are further elaborated on below with specific reference to the material in each reference that forms the basis for and/or supports Applicant's arguments.

**US Patent 5,706,514 (Bonola).**

Bonola describes a multiprocessor system of multiple "multimode" microprocessors. Each "multimode" microprocessor in Bonola would be best equated as an example of an elemental processing resource of the "first type" in the instant claims, irrespective of whether it is a master or slave. That is because, in Bonola, each microprocessor fetches and executes an instruction stream from memory, and inter-processor communication among the microprocessors is by software convention. [See, e.g. Bonola at: Fig 1; Col. 3, Lines 16-26 and 45-51; Col. 4, Lines 43-47; Col. 7, Lines 7-15; and Col. 8, Lines 55-61]

Bonola does not describe anything that would be an elemental processing resource of the claimed "second type" at all because instructions are not delegated by any multimode processor of Bonola to any other processor in Bonola for execution. [See, e.g. Bonola at: Col. 3, Lines 37-42 and 45-51; and Col. 8, Lines 55-61]

As a result, since Bonola does not have "second type" processing resources of the pending claims, Bonola cannot disclose or hint at any form of method involving delegating instructions to such a second type elemental processing resource, let alone a method that requires at least two first type elemental processing resources sharing at least two second type elemental processing resources.

**US Patent 6,081,860 (Bridges et al.)**

Bridges et al. describes a multiprocessor system made up of multiple microprocessors that connect to multiple memory controllers or devices via a shared, arbitrated Processor Local Bus (PLB). The bus arbiter can accept a memory request while another is in progress, pipelining the address of the next request so data transfer can begin immediately after the conclusion of the previous request. It refers to the processors as "master devices" and the memory controllers or devices as "slave devices", but it is a memory address that is pipelined and memory data that is transferred, so it is clear that the "slave" is not a slave processor as the Office Action asserts. [See, e.g. Bridges at: Figs 1-5; Col. 2, Lines 47-52; Col. 4, Lines 9-12 and 49-62]

Each "master device" processor in Bridges et al. would best be equated as an example of an elemental processing resource of the "first type." This is clear because the "master device" processors in Bridges et al. all originate all memory references. [See, e.g. Bridges at: Fig 1; Col. 2, Lines 52-57; Col. 4, Lines 28-30 and 36-41]

In contrast, as mentioned above, each "slave device" in Bridges et al. is a memory controller or device that does not process the information transferred, so is not an elemental processing resource of the "second type." Each such memory controller or device simply reads and writes memory in the usual ways but additionally incorporates address pipelining. [See, e.g. Bridges at: Fig 1; Col. 2, Lines 47-52; Col. 4, Lines 9-12 and 49-62]

Thus, Bridges et al. does not describe anything that would be an elemental processing resource of the "second type" of the claims at all. As a result, lacking such a "second type" processing resource, Bridges et al. does not, indeed cannot, disclose or hint at any form of method involving delegating instructions to a second type elemental processing resource, let

alone a method that requires at least two first type elemental processing resources sharing at least two second type elemental processing resources.

**US Patent 6,907,454 (Butterworth et al.)**

Butterworth et al. describes a dual processor system with a high performance master processor connected, via a bridge circuit, to a lower performance slave processor and various high and low speed memory busses.

The specific processors mentioned in Butterworth et al. are an IBM PowerPC 705 master processor and a PowerPC 403 slave processor that are connected to each other, and to memory, by a bus subsystem. It is therefore clear that the slave processor is not fundamentally different from the master processor, but is merely of lower performance. [See, e.g. Butterworth at: Fig 1; Col. 3, Lines 14-18 and 23-27; and Col. 4, Lines 29-33]

The high performance master processor uses software commands passed via a bus subsystem to the slave processor in order to delegate to software drivers on the slave processor access to memory and devices on the high and slow speed memory busses. [See, e.g. Butterworth at: Figs 4-5; Col. 3, Lines 14-18 and 23-27; Col. 4, Lines 43-48; Col. 5, Lines 7-9; and Col. 6, Lines 45-63]

Moreover, since Butterworth et al. is simply a dual processor system, it lacks the requisite multiple master processors which, by definition, are necessary for sharing of another processor between them.

Each processor in Butterworth et al. would best be equated as an example of a claimed elemental processing resource of a "first type," irrespective of whether it is a master or slave. This is clear because each processor in Butterworth et al. fetches and executes an instruction stream from memory and inter-processor communication is by software convention, with commands and data passed via a bus subsystem. [See, e.g. Butterworth at: Figs 1, 3-5; Col. 3, Lines 14-18; Col. 4, Lines 43-48; Col. 5, Lines 7-9; and Col. 6, Lines 45-63]

Thus, Butterworth et al. does not describe anything that would be an elemental processing resource of a second type at all. As a result, Butterworth et al. does not, indeed cannot, disclose or hint at any form of method involving delegating instructions to a second type elemental processing resource, let alone a method that requires at least two first type elemental processing resources sharing at least two second type elemental processing resources.

**US Patent 5,978,838 (Mohamed et al.)**

Mohamed et al. describes a dual processor system on a chip where the two processors execute different instruction sets. [See, e.g. Mohamed at: Fig 1; Col. 2, Lines 8-9 and 31-34; and Col. 3, Lines 32-35] One of the two processors is a general purpose processor and the other is an SIMD vector processor. [See, e.g. Mohamed at: Fig 1; Col. 2, Lines 31-34; Col. 3, Lines 32-35] Although, as a general matter, an SIMD vector processor could be used as an elemental processing resource of a second type of the claims, that is specifically not the case with Mohamed et al. because each processor in Mohamed has its own instruction and data caches (i.e. both processors are fully independent peer processors that each execute their own software from memory). [See, e.g. Mohamed at: Fig 1; Col. 1, Lines 61-67; Col. 2, Lines 66-67; Col. 3, Lines 1-2 and 37-43] All the two processors have in common is a main memory bus and interface registers. [See, e.g. Mohamed at: Fig 1 and Col. 3 Lines 15-20]

Moreover, since Mohamed et al. is simply a dual processor system without multiple master processors, necessarily there can be no sharing by two processors of any other processor(s).

Both processors in Mohamed et al. would best be equated as examples of an elemental processing resource of the claimed "first type," even though one is a general purpose processor and the other an SIMD vector processor. This is because, as noted above, each processor independently fetches and executes an instruction stream from memory and inter-processor communication is by software convention, with commands and data passed via registers and interrupt signals in bridge circuitry. [See, e.g. Mohamed at: Fig 1; Col. 1, Lines 61-67; Col. 2, Lines 66-67; Col. 3, Lines 1-2 and 15-20; and Col. 3 Lines 37-43]

Thus, Mohamed et al. does not describe anything that would be an elemental processing resource of a second type at all. As a result, Mohamed et al. does not, indeed cannot, disclose or hint at any form of method involving delegating instructions to a second type elemental processing resource, let alone a method that requires at least two first type elemental processing resources sharing at least two second type elemental processing resources.

**Claim 144 Is Allowable**

Based upon the foregoing, it should be appreciated that none of the cited references, taken alone or in any combination, disclose or suggest at any of the following aspects of independent claim 144, let alone any combination of such aspects:

- "the processing resources of the first type each being defined by a configuration whereby it will receive a non-delegated instruction from memory that it can execute or delegate for execution elsewhere, the processing resources of the second type each being defined by being configured to only receive and execute instructions that were delegated to them by processing resources of the first type and by being shared by at least two elemental processing resources of the first type"
- "two or more elemental processing resources of a first type each sharing two or more elemental processing resources of a second type with the at least one other of the two or more processing resources of the first type"
- "determining whether an operation-code within the execution instruction is incapable of being executed by the one elemental processing resource ofthe first type and thus should be delegated to one of the shared elemental processing resource of the second type"
- "if the execution instruction should be delegated, routing the execution instruction to a shared elemental processing resource of the second type, of the at least two of that type, that is capable of executing the operation code"

As further demonstrated above, all of the cited references lack any "second type" elemental processing resources as claimed. Thus, since no combination of the cited references disclose or suggest that aspect, let alone any one or more of at least the bulleted aspects, claim 144 is non-obvious and allowable.

**The Claims Depending From Claim 144 Are Allowable**

Claims 145, 155-158, 160, 164-165, and 167-172 all depend, directly or indirectly, from claim 144, so they are all allowable for at least the same reasons.

Moreover, the dependent claims add additional aspects that are lacking from the cited references as well, taken alone or in combination. Accordingly, those dependent claims are independently allowable.

For example, dependent claims 158, 160 and 164 respectively each specify at least one specific "flavor" for one or more of the second type processing resources. Since, as demonstrated above, the cited references do not disclose or suggest any "second type" processing resources as defined in the claims at all, they necessarily cannot disclose such particular types.

In addition, claims 158, 160 and 164 allow for both homogeneity or heterogeneity among the second type processing resources because they require "one or more" of the at least two to be of a particular "flavor" of resource. This aspect is similarly lacking from all of the references.

Dependent claim 167 requires, while acting on instructions from an individual thread, that a first type resource to sleep while a second type resource executes an instruction from the thread delegated to it. No such operation is remotely hinted at by the cited references.

Dependent claim 168 specifically requires the resources "dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput." The Office Action-cited part of Bonola does not begin to disclose or suggest this aspect. All Bonola discloses is that a processor can "sleep" a far cry from regulating sleep in the manner claimed.

Similarly, Bonola's simple disclosure of a processor that can sleep does not begin to disclose or suggest the subject matter of claims 169 and 170 which respectively require "generating an execution-instruction signal from at least one of the elemental processing resources, wherein the execution-instruction signal from the elemental processing resources themselves shuts off processing resources while idling" and "generating an execution-instruction signal from at least one of the elemental processing resources, wherein an execution-instruction signal from processing resources themselves turn on processing resources when execution instruction signal processing is required."

Accordingly, the above-referenced dependent claims are all additionally allowable for those independent reasons.

**Claim 1615 Is Allowable**

As should now be clear from the portions of the cited references identified by Applicant, none of the cited art, alone or in combination, disclose claim 1615 because they neither disclose nor suggest any of the following aspects of independent claim 1615 alone or in combination:

- "at least two IPU type processing resources"
- "two or more elemental processing resources of an other type comprising, in any combination, one or more of either an MPU type, an instruction delegating cache type or a vector type"
- " the elemental processing resources being configured such that a first and a second of theIPU type processing resources each share at least two of the other type elemental processing resources"

- "delegating the other execution-instruction from the thread to one of the two or more elemental processing resources of the other type that is shared between the first IPU and a second IPU"
- "maintaining the thread in the first IPU in a sleep state until an indicator that the processing of the other execution-instruction from the thread by the other type processing resource is returned"
- "determining that the execution-instruction of the different thread can not be processed by the second IPU but can be processed by the other type elemental processing resource that is shared between the second IPU and the first IPU"
- "delegating the execution-instruction from the different thread to the other type elemental processing resource that is shared between the second IPU and the first IPU"

Accordingly, claim 1615 would not have been obvious in view of any of the cited references.

## CONCLUSION

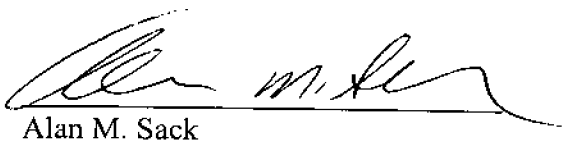
It is respectfully submitted that all of the pending claims would not have been obvious in view of the references cited by the Office Action, whether taken alone or in combination and the objections have been overcome. Accordingly, it is respectfully submitted that all of the pending claims are allowable and early, favorable action in that regard is respectfully requested.

## AUTHORIZATION

The Commissioner is hereby authorized to charge any additional fees which may be required for consideration of this Amendment to Deposit Account No. **504827**, Order No. **1004437-006US**.

Respectfully submitted,
Locke Lord Bissell & Liddell, L.L.P.

Dated: July 19, 2010

By: 
Alan M. Sack
Reg. No. 31,874

Correspondence Address:
Associated with Customer No. **85775**